Reason to use `⊢` : "Turnstile"! (You shall not pass, until you've given me -- that is, proved for me, this goal.)

To play with the geometry by hand, go to: https://www.geogebra.org/classic?lang=en

We're in the middle of the proof of `PropIp1` where we're trying to establish
`have Step3 : Circs_inter α β`
We'll remember that `have := Circs_inter_iff α β` says:

```
this : Circs_inter α β ↔ ∃ a b, Pt_on_Circ a α ∧ Pt_in_Circ a β ∧ Pt_on_Circ
b β ∧ Pt_in_Circ b α
```

And remember that `↔` is actually two things, a forward direction, and a backwards direction. The forward direction is `Circs_inter α β → ∃ a b, Pt_on_Circ a α ∧ Pt_in_Circ a β ∧ Pt_on_Circ b β ∧ Pt_in_Circ b α` which is only useful if we already know that `Circs_inter`. Only the backwards direction is useful to us. So we write:

```
have := (Circs_inter_iff α β).2
```

which gives us:

```
this : (∃ a b, Pt_on_Circ a α ∧ Pt_in_Circ a β ∧ Pt_on_Circ b β ∧ Pt_in_Circ
b α) → Circs_inter α β
```

One more tactic: If you are trying to prove `⊢ Q` and you know that `H : P → Q`, then if we `apply H`, then it will suffice to prove `P`.
In other words, if your goal state is:

```
H : P → Q
⊢ Q
```

and you write `apply H`, then the goal state will become:

```
⊢ P
```

In our setting, we have the goal state:

```
this : (∃ a b, Pt_on_Circ a α ∧ Pt_in_Circ a β ∧ Pt_on_Circ b β ∧
    Pt_in_Circ b α) → Circs_inter α β
⊢ Circs_inter α β
```

so when we write `apply this`, we get the new goal state:

```
⊢ ∃ a b, Pt_on_Circ a α ∧ Pt_in_Circ a β ∧ Pt_on_Circ b β ∧ Pt_in_Circ b α
```

When we see `∃` in the goal, we need to `use` something. What we actually need to use is `b` which is on `α` and `a` which is on `β`. After `use b, a`, the goal has become:

```
⊢ Pt_on_Circ b α ∧ Pt_in_Circ b β ∧ Pt_on_Circ a β ∧ Pt_in_Circ a α
```

And to break up this big goal into smaller goals, we write `constructor`. The outcome is:

```
⊢ Pt_on_Circ b α
⊢ Pt_in_Circ b β ∧ Pt_on_Circ a β ∧ Pt_in_Circ a α
```

We zero in on the first goal with `·`.
This is where we left off:

```
import Mathlib

class EuclideanPlane where
  Pt : Type
  Line : Type
  Circ : Type
  Pt_on_Line : Pt → Line → Prop
  Line_of_Pts : ∀ a b : Pt, ∃ L : Line, (Pt_on_Line a L) ∧ (Pt_on_Line b L)
  Pt_on_Circ : Pt → Circ → Prop
  Center_of_Circ : Pt → Circ → Prop
  Circ_of_Pts : ∀ a b : Pt, ∃ α : Circ, (Center_of_Circ a α) ∧ (Pt_on_Circ b
α)
  dist : Pt → Pt → ℝ
  dist_symm : ∀ (a b : Pt), dist a b = dist b a
  dist_nonneg : ∀ (a b : Pt), 0 ≤ dist a b
  dist_pos_def : ∀ (a b : Pt), dist a b = 0 ↔ a = b
  Circs_inter : Circ → Circ → Prop
  Pts_of_Circs_inter : ∀ α β : Circ, Circs_inter α β → ∃ a b : Pt,
```

```
    Pt_on_Circ a α ∧ Pt_on_Circ b α ∧ Pt_on_Circ a β ∧ Pt_on_Circ b β ∧ a ≠
b
  Pt_in_Circ : Pt → Circ → Prop
  Circs_inter_iff : ∀ (α β : Circ), Circs_inter α β ↔
    (∃ (a b : Pt), Pt_on_Circ a α ∧ Pt_in_Circ a β ∧
      Pt_on_Circ b β ∧ Pt_in_Circ b α)
  Pt_on_Circ_iff : ∀ (a b c : Pt) (α : Circ), Center_of_Circ a α →
    Pt_on_Circ b α → (Pt_on_Circ c α ↔ dist a c = dist a b)
  Pt_in_Circ_iff : ∀ (a b c : Pt) (α : Circ), Center_of_Circ a α →
    Pt_on_Circ b α → (Pt_in_Circ c α ↔ dist a c < dist a b)

variable [EuclideanPlane]

namespace EuclideanPlane

def IsEquilateralTriangle (a b c : Pt) : Prop := (dist a b = dist b c) ∧
(dist a b = dist a c)

-- Let's make a helper lemma: given a circle `α` and a point `a` that's the
center of the circle, `a` is in the interior of the circle
lemma in_Circ_of_center (a : Pt) (α : Circ) (a_cent_α : Center_of_Circ a α)
:
    Pt_in_Circ a α := by
  sorry -- HOMEWORK

theorem PropIp1 (a b : Pt) : ∃ (c : Pt), IsEquilateralTriangle a b c := by
  have Step1 : ∃ (α : Circ), Center_of_Circ a α ∧ Pt_on_Circ b α :=
    Circ_of_Pts a b
  obtain ⟨α, a_center_of_α, b_on_α⟩ := Step1
  have Step2 : ∃ (β : Circ), Center_of_Circ b β ∧ Pt_on_Circ a β :=
    Circ_of_Pts b a
  obtain ⟨β, b_center_of_β, a_on_β⟩ := Step2
  have Step3 : Circs_inter α β := by
    have := (Circs_inter_iff α β).2
    apply this
    use b, a
    constructor
    · exact b_on_α
    · constructor
      ·
```

```
        have := (Pt_in_Circ_iff b a b β b_center_of_β a_on_β).2

        sorry -- Pt_in_Circ b β
      · constructor
        · exact a_on_β
        · sorry -- Pt_in_Circ a α
  have Step4 : ∃ (c d : Pt), Pt_on_Circ c α ∧ Pt_on_Circ d α ∧
    Pt_on_Circ c β ∧ Pt_on_Circ d β ∧ c ≠ d :=
      Pts_of_Circs_inter α β Step3
  obtain ⟨c, d, c_on_α, d_on_α, c_on_β, d_on_β, c_ne_d⟩ := Step4
  -- next step: create a line through `a` and `c`
  --have Step5 : ∃ (L : Line), Pt_on_Line a L ∧ Pt_on_Line c L :=
Line_of_Pts a c
  --obtain ⟨L, a_on_L, c_on_L⟩ := Step5
  have Step5 : dist a c = dist a b :=
    (Pt_on_Circ_iff a b c α a_center_of_α b_on_α).1 c_on_α

  have Step6 : dist b a = dist b c :=
    (Pt_on_Circ_iff b c a β b_center_of_β c_on_β).1 a_on_β

  use c
  unfold IsEquilateralTriangle
  constructor
  · convert Step6 using 1
    have := dist_symm a b
    --have := Step5.symm
    exact this
  ·  --- HOMEWORK
    --exact Step5.symm
    have Step7 := Step5.symm
    exact Step7
```